

Remarks

Status of application

Claims 1-63 were examined and stand rejected in view of prior art and claims 1, 21, 23 and 62 have also been objected to for technical reasons. The claims have been amended to further clarify Applicant's invention. In view of the amendments made and the following remarks, reexamination and reconsideration are respectfully requested.

General

A. Objections to Claims

The Examiner has objected to claims 1 and 23 as containing informalities. Applicant has amended these claims as suggested by the Examiner, thereby overcoming the objection.

B. Section 112, second paragraph rejection

Claims 21 and 62 stand rejected under 35 U.S.C. 112, second paragraph as being indefinite. Applicant has amended claims 21 and 62 so as to overcome this rejection.

Prior Art Rejections

A. First Section 103 Rejection: Goodwin and Barrett

Claims 1-7, 12-29, 34, 37-48, 53 and 56-63 stand rejected under 35 U.S.C. Section 103(a) as being unpatentable over U.S. Patent 6,199,195 to Goodwin et al (hereinafter "Goodwin") in view of U.S. Patent 7,000,219 to Barrett et al (hereinafter "Barrett"). The rejection of Applicant's claim 1 as follows is representative of the Examiner's rejection of Applicant's claims as unpatentable over Goodwin and Barrett:

As per Claim 1, Goodwin discloses:
creating a model describing business objects and rules of the application (see Column 6: 37-44, "The system and method will also allow developers to generate objects based on a framework of services they author by composing services based on the object templates into objects that support the composed behaviors and methods. This is accomplished through the code generator 210, which interfaces with the unified models 206, which are expressed in a unified modeling language, such as Unified Modeling Language (UML).");

creating source code for the application (see Column 13: 52-55, "Thus, the code generator 330 can support the creation of for example, IDL, JAVA or C++ files ...");

compiling the source code into an executable application (see Column 14: 34-40, "Outputs from the code generator 404 include Interface Definition Language (IDL) files 422b (*. idl), which are processed by an idl-to-Java compiler, e.g., Visibroker's IDL2JAVA, for the generation of CORBA ORB services ..."); and running the executable application on a target computer in conjunction with a runtime framework that provides services to the executable application (see Column 15: 45-65, "The data server 332 operates at run time ... " and "When deployed within a client application, the data server 332 launches, starts, manages and controls execution of a set of services.").

However, Goodwin does not disclose:

representing the model within the source code itself; and while the executable application is running, reconstructing the model from the executable application and making it available to the run-time framework.

Barrett discloses:

representing the model within the source code itself (see Column 5: 48-52, "In the requirements and specification phases 1 and 2 the system design is modelled. In the preferred embodiment, this model comprises a series of UML diagrams. The model is developed by the UML tool 15 and recorded in a file in an XMI/XML form."); and

while the executable application is running, reconstructing the model from the executable application and making it available to the run-time framework (see Column 5: 37-44, "During execution of a system, the RAS 10 maintains the meta-model associated with the executing system. During this phase, the meta model remains accessible to the UML tool. Changes made to the meta model have immediate effect on the executing system by addition, removal and/or replacement of components, and changes to the structure of the bindings between instances according to changes to the meta model.").

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Barrett into the teaching of Goodwin to include representing the model within the source code itself; and while the executable application is running, reconstructing the model from the executable application and making it available to the run-time framework. The modification would be obvious because one of ordinary skill in the art would be motivated to maintain a "flexible" software system (see Barrett - Column 1: 11-16).

As illustrated above, the Examiner continues to rely on Goodwin as substantially teaching Applicant's claimed invention. However, Applicant respectfully believes that Applicant's invention is distinguishable from Goodwin in a number of respects as described in detail in Applicant's previously filed Appeal Brief (which is hereby incorporated by reference). To briefly recap, Applicant's invention does not simply

generate application code based on a model (e.g., UML model) like Goodwin's solution. Instead, Applicant's invention represents the UML model inside the source code of the application itself so as to enable the model to be reconstructed from the executable application code at runtime. Applicant's invention ensures that everything in the model is represented in the emitted source code, so that it can later be reconstructed from the code to a model again (Applicant's specification, paragraphs [0060] - [0061]). This includes not only representing portions of the UML model having a natural representation in the source code (e.g., class names), but also involves adding attributes and other information to the code for conveying meta information about the UML model which is not otherwise expressible in the code (Applicant's specification, paragraph [0070]; Fig. 3D). Thus, Applicant's claimed invention incorporates a full representation of the model into the source code itself. This enables the entire model that was used for generating the code to be fully reconstructed back into model form at runtime from the executable code itself as hereinafter described.

The Examiner now acknowledges that Goodwin does not include these features of representing the model within the source code itself and reconstructing the model from the executable application while the application is running and, therefore, adds Barrett as providing these teachings. However, turning to Barrett, one finds that Barrett does not represent the model in source code, but rather creates a meta model of the UML model and stores this meta model as an XMI file. Additionally, Barrett's solution does not actually reconstruct the model from the executable application, but rather from a meta model that is associated with the system that is executing at runtime. These and other differences are described below in further detail.

Barrett's solution provides for transforming a UML model into a meta model having a hierarchical structure defining how components of a software system are to be instantiated at runtime (Barrett, Abstract; col. 5, lines 8-13). A runtime architecture service (RAS) of Barrett's solution can then dynamically assemble a system in runtime according to the meta model (Barrett, Abstract). More particularly, the RAS assembles the system by instantiating, loading and binding components drawn from a component set to create a software system structured as a set of components configured so that the overall system architecture matches the design contained in the meta model (Barrett, col.

5, lines 14-25).

Significantly, however, Barrett's solution loads and saves models as XMI files as follows:

Referring to FIG. 3, the RAS 10 loads and saves UML models 3 as XMI files. As stated above, the RAS 10 generates a meta model from each UML model 3. Each meta model is a hierarchical structure, and may be edited/manipulated by a UML tool 15 via an API. This is performed in a user-friendly manner as the tool 15 provides a view on the meta model, and provides methods for opening, closing, and type correct manipulation. When the RAS 10 saves a meta model it takes the form of the corresponding UML model 3 in an XMI file. The meta model may thus be regarded as an object representation of the UML model 3. During execution of a system, the RAS 10 maintains the meta-model associated with the executing system. During this phase, the meta model remains accessible to the UML tool. Changes made to the meta model have immediate effect on the executing system by addition, removal and/or replacement of components, and changes to the structure of the bindings between instances according to changes to the meta model.

(Barrett, col. 5, lines 26-44, emphasis added).

As noted above, Barrett's solution transforms a UML model into a meta model which it then saves as an XMI file. XML Metadata Interchange (XMI) is an Object Management Group (OMG) standard for exchanging meta data information via Extensible Markup Language (XML). It provides a mapping from UML (or another model) to XML. Importantly, this XMI information is stored in a separate file and Barrett makes no mention of representing the UML model within the source code itself. Applicant's claimed invention, in contrast, specifically calls for representing the model within the source code itself as source code and code attributes (Applicant's specification, paragraph [0061]). To further clarify that Applicant's invention represents the model in the code, Applicant has amended its independent claims to specifically indicate that the model is represented in the source code as source code and code attributes. For example, Applicant's amended claim 1 includes the following claim limitations:

In a computer system, an improved method for developing and executing an application, the improved method comprising:
creating a model describing business objects and rules of the application;

creating source code for the application, including representing the model within the source code itself, wherein the model is represented as source code and code attributes;
compiling the source code into an executable application;
running the executable application on a target computer in conjunction with a run-time framework that provides services to the executable application; and
while the executable application is running, reconstructing the model from the executable application and making it available to the run-time framework.

(Applicant's amended claim 1, emphasis added)

Applicant's invention provides for representing model information in the source code itself. More particularly, the model information is expressed in the code as "normal" source code and code attributes (Applicant's specification, paragraphs [0060]-[0061]). In contrast, Barrett's system stores model information in XMI format in a separate file rather than expressed in the application's source code.

Another difference between Applicant's claimed invention and Barrett's solution is that Barrett reconstructs the runtime model from this stored XMI-representation and not from a model represented in code of the executable application itself. As noted above, Barrett's solution "maintains the meta-model associated with the executing system" (Barrett, col. 5, lines 37-38). Moreover, the meta-model remains available to the UML tool and changes to the model have immediate effect on the executing system by addition, removal and/or replacement of system components (Barrett, col. 5, lines 39-44). These features make it obvious that Barrett's system does not reconstruct the model from compiled code since it can be changed in the UML tool while the system is running. Thus, it is clear that the model (in Barrett) is not actually reconstructed from the executable application, but rather from a "meta-model associated with the executing system" stored in a separate file. Additionally, Barrett's approach of relying on a stored XMI representation may result in version conflict as the software and the separate stored representation may change such that the stored model does not reflect the actual code implementation. Applicant's solution is specifically designed to avoid this problem of version mismatch by storing the model information inside the code itself (Applicant's specification, paragraph [0059]). Accordingly, Barrett's approach is not comparable to Applicant's claimed invention as it does not provide for representing the model in the source code itself nor does it teach reconstructing the model from the executable

application.

Further distinctions between Applicant's solution and the prior art references are also found in Applicant's dependent claims. For example, as to Applicant's claim 7 the Examiner states that it is well known in the art to use reflection technique to read metadata. Although Applicant acknowledges that the general teaching of reflection technique is known in the art, Applicant's solution provides for use of reflection technique to reconstruct a model (e.g., UML model) from information which is incorporated into the source code of an application when the application was developed. Applicant's approach provides that a compiled application is inspected using introspection/reflection mechanisms to reconstruct the model from the information expressed in the code (Applicant's specification, paragraphs [0061] and [0166]). Applicant has amended claims 7, 29 and 48 to clarify that reflection is used to read information incorporated into the executable application. Barrett, in contrast, maintains a meta-model in a separate stored XMI file as discussed above. Applicant further believes that its use of reflection to reconstruct a UML model at runtime from information incorporated into the application itself is a novel use of introspection/reflection mechanisms not previously known in the art.

As another example, Applicants claims 12, 34 and 53 include claim limitations of testing integrity of the reconstructed model after the model is reconstructed at runtime. The Examiner references the following portion of Goodwin as including the equivalent teachings:

Any authenticated utility can access the meta-information through the schema server 316 using a queryable interface and/or an Interface Definition Language (IDL) interface of a unified modeling language model (unified model 206 (Fig. 2)).

(Goodwin, col. 10, lines 7-11)

The above-referenced portion of Goodwin simply describes a interface to the model meta-information through the schema server. It is not at all analogous to Applicant's claimed limitations of testing integrity of the model after it is reconstructed.

All told, Goodwin and Barrett, even when combined, provide no teaching of fully representing the model inside the source code itself. The representation of the model in

the source code is essential to the process of reconstructing the model from the executable application at runtime. Thus, neither Goodwin's nor Barrett's solution can reconstruct a model from the executable application when the model is not represented in the source code of the application in the first place. As described above, Barrett stores a meta-model in a separate file and does not reconstruct the model from the executable application at runtime in the manner described in Applicant's specification and claims. As the combined references do not teach or suggest all of the claim limitations of Applicant's claims 1-7, 12-29, 34, 37-48, 53 and 56-63 (and other claims) it is respectfully submitted that the claims distinguish over the references and overcome the Section 103 rejection.

B. Second Section 103 Rejection: Goodwin, Barrett and Moore

Claims 8, 10, 11, 30, 32, 33, 49, 51 and 52 stand rejected under 35 U.S.C. Section 103(a) as being unpatentable over Goodwin (above), in view of Barrett (above), further in view of U.S. Patent 6,560,769 to Moore et al (hereinafter "Moore"). As to these claims, Applicant believes that the claims are allowable for at least the reasons set forth above in the response to the first Section 103 rejection as to the deficiencies of the Goodwin and Barrett references with respect to Applicant's invention. In particular, neither Goodwin nor Barrett provides for fully representing the model within the application source code itself and therefore cannot reconstruct the model from the application. This is further confirmed by the fact that the Examiner acknowledges that Goodwin and Barrett do not disclose the features of these dependent claims which discuss reconstructing portions of the model from code elements that are encountered (Claims 8, 30 and 49), detecting a class having a package element and creating a corresponding UML package (Claims 10, 32 and 51), or detecting a attribute indicated a class belongs to a UML package and specifying in the UML model that the class belongs to that UML package (Claims 11, 33 and 52).

Moore does not cure any of the deficiencies of the prior art references as to Applicant's claimed invention. Moore describes a system which parses Java source code files in order to generate a UML representation of such code (Moore, Abstract, Fig. 2, col. 4, lines 19-26). Applicant's claimed invention, in contrast, does not create a UML

representation from source code files. Rather, Applicant's claimed invention provides for reconstructing a model from a compiled, executable application using introspection/reflection techniques. In this regard, Applicant notes that these dependent claims must be read in conjunction with the underlying independent claims (see e.g., claim 1) providing that the UML model is reconstructed from an executable application as well as intervening claims (see e.g., claim 7) which provides for use of reflection techniques. Moore makes no mention of using reflection techniques, but instead provides for generating UML model information by parsing source code files. Respectfully, this is not comparable to Applicant's claimed invention. Therefore, as the prior art references, even when combined, do not teach or suggest all of the claim limitations of Applicant's claims 8, 10, 11, 30, 32, 33, 49, 51 and 52 it is respectfully submitted that the claims distinguish over the prior art references and overcome any rejection under Section 103.

C. Third Section 103 rejection: Goodwin, Barrett and Schloegel

Claims 9, 31, and 50 stand rejected under 35 U.S.C. Section 103(a) as being unpatentable over Goodwin (above) in view of Barrett (above), and further in view of US Published Application 2004/0044990 of Schloegel (hereinafter "Schloegel"). As to these claims, the Examiner acknowledges that Goodwin and Mullins do not disclose spanning the graph using a selected one of depth-first, breadth-first and ad-hoc traversal techniques and thus adds Schloegel for these teachings.

Claims 9, 31 and 50 are dependent upon Applicant's independent claims 1, 23 and 43 as well as dependent claims 7, 29 and 48 and, therefore, are believed to be allowable for at least the reasons cited above pertaining to the above-described deficiencies of Goodwin and Barrett in respect to Applicant's invention. As discussed in Applicant's previously filed Appeal Brief, Schloegel does not cure these deficiencies as Schloegel simply describes alternative techniques for traversal during code generation as follows:

The order of entity traversal during code generation can be application specific. For example, the order of entity traversal during code generation could be random, or ordered by type, or sorted by name, or filtered so that only entities with a specific property are traversed, or the graph could be traversed in various other ways such as depth-first traversal.

(Schloegel, paragraph [0033], emphasis added).

As shown above, Schloegel discusses traversal during code generation, while Applicant's claimed invention creates a graph of code elements based on the executable application itself and then spans the graph using depth-first, breadth-first or ad-hoc traversal techniques to reconstruct the model from the executable application. As Schloegel does not include any teaching of representing a unified model within application code generated from the model or of reconstructing the model from the executable application at runtime, it does not cure any of the above-described deficiencies of Goodwin and Barrett as to Applicant's invention. Therefore, as the combined references do not teach or suggest all the claimed features of Applicant's invention, it is respectfully submitted that Applicant's claimed invention as set forth by these claims is distinguishable over the references and overcomes the rejection under Section 103.

D. Fourth Section 103 rejection: Goodwin, Barrett and Mutschler

Claims 13, 14, 35, 36, 54 and 55 stand rejected under Section 103(a) as being unpatentable over Goodwin (above) in view of Barrett (above), further in view of U.S. Patent 7,162,462 to Mutschler III (hereinafter "Mutschler"). The Examiner again relies on Goodwin and Barrett as substantially teaching Applicant's claimed invention, but acknowledges that these references do not disclose constructing a common superclass for classes of the reconstructed model. Therefore, Mutschler is added as providing this teaching.

Claims 13, 14, 35, 36, 54 and 55 are dependent upon Applicant's independent claims 1, 23 and 43 as well as dependent claims 12, 34 and 53 and, therefore, are believed to be allowable for at least the reasons set forth above as to the deficiencies of Goodwin and Barrett with respect to Applicant's invention. Mutschler does not cure any of these deficiencies as it simply describes the creation of a superclass for both events and rules, allowing them to share data and methods relating to their common dynamic behavior over time (Mutschler, col. 5, lines 40-43) in the context of a system providing time sensitivity to an inference engine (Mutschler, Abstract). Applicant's claimed invention, in contrast, ensures that all classes in the model belong to a common

superclass which may include constructing a common superclass if it is found that all classes in the reconstructed model do not share a common superclass (Applicant's Fig. 5B at 513; paragraphs [172]-[174]). Applicant's invention performs these steps during the process of traversing the reconstructed model information to ensure integrity of the model (Applicant's Fig. 5B at 513). The prior art references do not include teachings of performing this type of integrity testing of a model by ensuring all classes in the model belong to a common superclass.

The Goodwin, Barrett and Mutschler references, even when combined, do not teach or suggest all the claimed features of Applicant's invention. Accordingly, it is respectfully submitted that Applicant's claimed invention is distinguishable over the combined references, and that the rejection of claims 13, 14, 35, 36, 54 and 55 under Section 103 is overcome.

Any dependent claims not explicitly discussed are believed to be allowable by virtue of dependency from Applicant's independent claims, as discussed in detail above.

Conclusion

In view of the foregoing remarks and the amendment to the claims, it is believed that all claims are now in condition for allowance. Hence, it is respectfully requested that the application be passed to issue at an early date.

If for any reason the Examiner feels that a telephone conference would in any way expedite prosecution of the subject application, the Examiner is invited to telephone the undersigned at 925 465-0361.

Respectfully submitted,

Date: April 18, 2008

/G. Mack Riddle/

G. Mack Riddle, Reg. No. 55,572
Attorney of Record

925 465-0361
925 465-8143 FAX